Atacama Pathfinder EXperiment

User Manual

APEX-MPI-MAN-0011
Revision: 1.14
Release: June 8, 2007
Category: 4
Author: Dirk Muders

APECS User Manual

Dirk Muders

Keywords: APECS, Observing, Operating

Author Signature: Dirk Muders	Date: June 8, 2007
Approved by: L.-Å. Nyman	Signature: L.-Å. Nyman
Institute: APEX	Date: June 11, 2007
Released by: L.-Å. Nyman	Signature: L.-Å. Nyman
Institute: APEX	Date: June 11, 2007

Change Record

Revision	Date	Author	Section/ Page affected	Remarks
1.0	2005-08-26	Dirk Muders	All	Initial Version
1.1	2006-06-16	Dirk Muders	Selected paragraphs	Updated to APECS 0.9.
1.11	2006-06-22	Dirk Muders	14	Updated FLASH frequency ranges.
			18	Updated "calibrate" command syntax.
1.12	2006-09-03	Dirk Muders	Selected paragraphs	Updated to APECS 1.0:
				Added spiral mode setup commands.
				Updated "use_ref" documentation.
				Fixed "derotate" command description.
1.13	2007-04-03	Dirk Muders	Introduction	Updated project directory structure.
			apecs commands	Added "drift" command. Iterated "project_id" and "spiral_setup" commands.
				Updated instrument list.
			CORBA CDB reference	Updated CORBA component list.
1.14	2007-06-08	Dirk Muders	apecs commands	Updated "Heterodyne frontend name", "source", "raster" and "otf" command syntax.
				Added "hexa" command.

Contents

1 What's new ? 7
1.1 Revision 1.13 . . . 7
1.2 Revision 1.12 . . . 7

2 Overview 8

3 Observing with the APEX Telescope 10
3.1 Introduction . . . 10
3.2 apecs commands . . . 11
3.2.1 General . . . 12
3.2.1.1 project_id . . . 12
3.2.1.2 operator_id . . . 12
3.2.1.3 observer_id . . . 12
3.2.1.4 show . . . 13
3.2.1.5 go . . . 13
3.2.1.6 cancel . . . 13
3.2.1.7 save_history . . . 13
3.2.1.8 run_macro . . . 13
3.2.1.9 save_defaults . . . 13
3.2.1.10 load_defaults . . . 13
3.2.1.11 reset_defaults . . . 13
3.2.1.12 observe . . . 14
3.2.1.13 continuous_data . . . 14
3.2.1.14 skip_hardware_setup . . . 14
3.2.2 Catalogs . . . 14
3.2.2.1 sourcecats . . . 14
3.2.2.2 ephemerides . . . 15
3.2.2.3 linecats . . . 15
3.2.3 Instruments . . . 15
3.2.3.1 frontends . . . 15
3.2.3.2 <Frontend name>.feeds . . . 16
3.2.3.3 <Continuum frontend name> . . . 17
3.2.3.4 <Heterodyne frontend name> . . . 17
3.2.3.5 <Heterodyne frontend name>.line . . . 17
3.2.3.6 <Heterodyne frontend name>.derotate . . . 18
3.2.3.7 <Frontend name>.backends . . . 18
3.2.3.8 <Continuum backend name> . . . 18
3.2.3.9 <Spectral backend name> . . . 18
3.2.3.10 <Spectral backend name>_group . . . 18
3.2.4 Target . . . 19
3.2.4.1 source . . . 19
3.2.5 Calibration . . . 19
3.2.5.1 calibrate . . . 19
3.2.5.2 skydip . . . 19

3.2.5.3 point . . . 20
3.2.5.4 pcorr . . . 20
3.2.5.5 pcorr_reset . . . 20
3.2.5.6 focus . . . 20
3.2.5.7 fcorr . . . 20
3.2.5.8 fcorr_reset . . . 21
3.2.5.9 set_tilts . . . 21
3.2.5.10 reset_tilts . . . 21
3.2.5.11 use_linear_sensors . . . 21
3.2.6 Observing Patterns . . . 21
3.2.6.1 offset . . . 21
3.2.6.2 reference . . . 22
3.2.6.3 use_ref . . . 22
3.2.6.4 on . . . 22
3.2.6.5 raster . . . 22
3.2.6.6 hexa . . . 23
3.2.6.7 otf . . . 23
3.2.6.8 drift . . . 23
3.2.6.9 spiral_setup . . . 24
3.2.6.10 use_spiral . . . 24
3.2.6.11 repeat . . . 24
3.2.7 Switch Mode . . . 24
3.2.7.1 tp . . . 24
3.2.7.2 wob . . . 25
3.2.7.3 fsw . . . 25
3.2.8 Antenna . . . 25
3.2.8.1 tolerance . . . 25
3.2.8.2 park . . . 25
3.2.8.3 stow . . . 25
3.2.8.4 unstow . . . 25
3.3 A typical Observing Session . . . 26
3.3.1 Source Setup . . . 26
3.3.2 Continuum Instrument Setup . . . 26
3.3.3 Spectral Line Instrument Setup . . . 26
3.3.4 Initial Calibrations . . . 26
3.3.5 Continuum Observations . . . 27
3.3.6 Spectral Line Observations . . . 27
3.4 Macros and Loops . . . 27
3.5 User defined Commands . . . 27
3.6 Notes and Caveats . . . 28

4 Operating the APEX Telescope 29
4.1 Introduction . . . 29
4.2 Starting the APECS servers . . . 29
4.3 Configurations . . . 30
4.4 Troubleshooting . . . 30
4.5 Notes and Caveats . . . 31

5 CORBA CDB Reference 37

List of Figures

2.1 APECS Pipeline Structure 8

4.1 APECS deployment at the site in Chile. 32

List of Tables

3.1 Example of an APEX source catalog. 14
3.2 Example of an APEX line catalog. 15
3.3 APEX heterodyne and continuum frontends. The receiver cabins are named A (left Nasmyth), B (right Nasmyth) and C (Cassegrain). 16
3.4 APEX spectral and continuum backends 16
3.5 Possible FEBE combinations for the APEX heterodyne frontends. 17
3.6 Possible FEBE combinations for the APEX bolometer frontends. 17

4.1 APECS hosts and standard processes. 33
4.2 Embedded system hosts and standard processes. 34
4.3 Known APECS problems and their solutions. 35
4.4 Known APECS problems and their solutions (ctd.). 36

5.1 APECS CORBA Objects and their activators. 37
5.2 APECS CORBA Objects and their activators (ctd.). 38
5.3 APECS CORBA Objects and their activators (ctd.). 39

Chapter 1

What's new ?

1.1 Revision 1.13

This minor revision includes the description of the "drift" command which is mainly used by the SZ project. The project_id and spiral_setup commands have been iterated. In addition, the project directory structure and the list of components have been updated to reflect the current status.

1.2 Revision 1.12

This revision describes APECS 1.0 as of September 3^{rd}, 2006. The main changes are the introduction of the spiral observing modes for bolometer observations and changing the mode parameter of the `use_ref` command to accept values 'on' and 'off' rather than 1 and 0 to homogenize the commands for various flag settings withing `apecs`.

Chapter 2

Overview

The Atacama Pathfinder EXperiment (APEX) Telescope is controlled by the "APEX Control System" (APECS). APECS is based on the "ALMA Common Software" (ACS) and the "(ALMA) Test Interferometer Control Software" (TICS). ACS provides the CORBA-based middleware communication layer to interface the hardware components to the control system. TICS provides the basic CORBA objects for antenna control in horizontal and equatorial coordinates. In addition to that, there are utilities to record several kinds of time stamped monitor points into a database (DB2) and to perform optical pointing runs. The ACS and TICS packages fulfill the requirements of common network communication, automatic monitoring, real-time tracking and remote observing. The overarching software to use all hardware devices in a coordinated way necessary for astronomical observations was developed by the APEX software development group ([3]). This included defining the instrument and device interfaces ([4], [2]) and the raw data format interface (MBFITS, [5]).

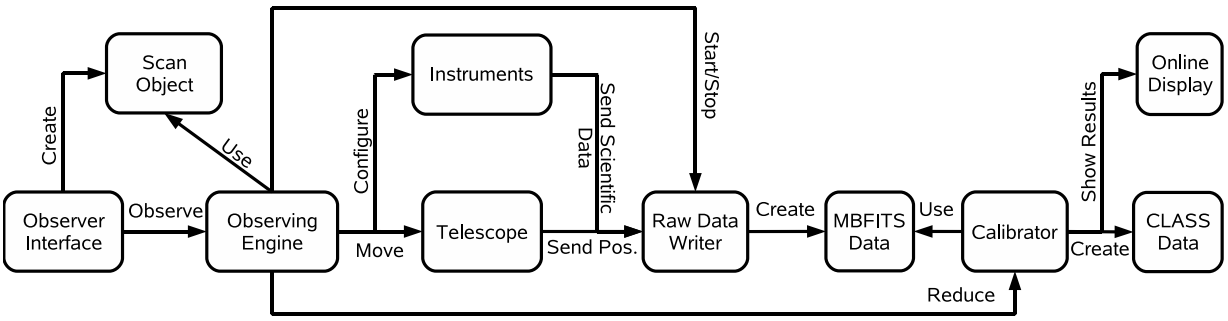

Figure 2.1: *This diagram illustrates the APECS observing pipeline structure. The astronomer submits a request for a scan – encoded as a so-called Scan Object – to the Observing Engine which then coordinates all hardware and software tasks to perform the observation. It sets up the instruments, moves the telescope to the desired position and starts the data recording. The Raw Data Writer collects the data streams and creates an MBFITS file. After each subscan the Calibrator provides calibrated data and shows results on the online display for user feedback.*

The interfaces for common instruments such as frontends or continuum and spectral backends are kept generic so that new instruments can be easily added to the system. The CORBA side of the interfaces is automatically generated from the interface files. Simulators can be used to run a full control system under Linux without the need for real hardware.
The observer level of APECS provides a scripting language for observing ("`apecs`") (a graphical user interface ("`xapecs`") is under development), the central Observing Engine to coordinate all devices and processes ("`apexObsEngine`"), the online MBFITS raw data writer ("`apexOnlineFitsWriter`") and the online data calibrator ("`apexOnlineCalibrator`") program to automatically perform the atmospheric corrections and provide CLASS (Gildas software) data with the T_A^* temperature scale ([7]). All scans are automatically logged as XML and HTML files using the `apexObsLoggerServer`.
In addition, there are a number of monitoring tools to see the antenna positions ("`apexAntMount`"), the Sun avoidance zone ("`plotAvoidance`"), the scan status ("`scanStatus`") and the values and possible

alarm states of all available monitor points ("`apexStatusDisplay masterStatus.MasterStatus`").
These APECS core components are organised as a pipeline system (see fig. 2.1). Observations are defined using so-called Scan Objects which contain the full description of the next observation, i.e. the instrument setup details, target coordinate information and the desired observing patterns. The Scan Objects, that are created by the observer command line interface, are sent to the Observing Engine which sets up all necessary devices, controls the data acquisition and triggers the online data calibration, reduction and display.
APECS applies a relativistic Doppler correction to spectral line data based on the SLALIB library as of 2003. We checked the velocities against the Stumpff library used at the Effelsberg telescope. The two libraries agree to within 0.03 km/s. We chose SLALIB because of its slightly more accurate algorithms taking e.g. lunar effects into account.
Spectral line data is being calibrated to the T_A^* temperature scale using Juan Pardo's ([6]) atmospheric library ATM. The calculations are made using Planck temperatures since the Rayleigh-Jeans approximation fails in the submm regime. Note, however, that the CLASS temperature scale is using the Rayleigh-Jeans approximation for compatibility reasons.

Chapter 3

Observing with the APEX Telescope

3.1 Introduction

Observations are performed using the APECS client computers, either in Sequitor (e.g. `pular` or `tacora`) or on Chajnantor (e.g. `toco` or `leon`). Note that one **must not** use any of the APECS server computers (`control`, `instruments` or `display`) for observations as their CPU resources are needed by the APECS services. One also **must not** run any other CPU intensive programs such as browsers, web cam displays, etc. on the servers. All APECS computers run under Linux (currently RedHat 7.2 due to the use of ACS 2.0.1).
The APECS startup procedures have been split into one server and two client scripts. The server script (`restartAPECSServers`) is started once on `control` and brings up all server programs necessary for observing. This part is done by the APEX operators or APEX astronomers on duty using the `apex` operations account. Note that currently the servers still need to be started within a VNC server (typically located at `control:1` to allow the observers to view the online data display (we are working on separating the Online Calibrator into a client/server pair to remove this restriction). After that server startup, clients can use the APECS services. It is possible to run several client sessions in parallel for different projects.
The operators will also create a new account for you based on your project ID. You must use that account for observing since all data products and log files are produced in separate project directories which are only accessible to the project account.
After logging on to one of the APECS client computers, you will find six symbolic links to your project's data:

- The `rawdata` link points to the directory containing the current `MBFITS` raw datasets on a computer at Chajnantor. The MBFITS dataset directories are named `APEX-<Scan number>-<Project ID>` where the scan number is a sequential number starting with "1" on January 1st every year and the project ID is defined as described in section 3.2.1.1.
- The `archived_rawdata` link points to the project's `MBFITS` raw datasets in the archive area on a computer in Sequitor. The data is copied to this location twice per day and one should use it from here when reducing data offline after the observations.
- The `scidata` link points to the directory containing the project's calibrated CLASS files at Chajnantor. The file naming scheme is `<Project ID>-<Date (as YYYY-MM-DD)>.apex`, i.e. you will find one file per day. The spectra are identified using the CLASS `TELESCOPE` header variable. Due to the variable length restriction of 12 characters, we had to use acronyms to specify the frontend and backend names and the pixel and baseband numbers as defined in MBFITS ([4]). The spectra naming scheme is `AP-<2-letter frontend acronym><2-digit pixel number>-<2-letter backend acronym><2-digit baseband number>`. The frontend and backend acronyms are given in tables 3.3 and 3.4.
- The `archived_scidata` link points to the project's calibrated CLASS files in the archive area on a computer in Sequitor. The data is copied to this location twice per day and one should use it from here when reducing data offline after the observations.
- The `obslogs` link points to the directory containing the project's automatically generated observation logs in XML and HTML format by the `apexObsLoggerServer`. The

files are named APECS-<Project ID>-<Date (as YYYY-MM-DD)>.obslog and APECS-<Project ID>-<Date (as YYYY-MM-DD)>.html. The apexObsLoggerClient application will display today's project specific logs in a window and allow editing the user comments while observing. System comments are given in square brackets. A file menu allows to view older logs of the same project. The log files are also transferred to the project database for remote access. The logs are part of the final data package that the PI will receive.

– The archived_obslogs link points to the project's automatically generated observation logs in the archive area on a computer in Sequitor. The logs are copied to this location twice per day and one should use them from here when reducing data offline after the observations.

Note that all of the above directories and files belong to the apex operations account. You cannot modify any of those files or create new ones within those directories as they are the base of the APEX archive. Make new directories and files off of your home directory and work from there. Please do *not* start applications inside the system directories since you do not have write permission there.
Apart from the data directories, there are a number of init files for the GILDAS software and the lpoint.class macro to reduce line pointings (type @lpoint in your CLASS session and enter the parameters asked for by the script; the line pointing fit will later be done automatically by the Online Calibrator).
If you log into a full KDE session, there is also a Desktop setup with the APECS_XTerm" icon. Note that the normal KDE terminals need to be started with the -ls option to correctly load the APECS paths.
To start your observing session type restartAPECSMonClients in one window. This will start several monitoring programs. Then type restartAPECSObsClients to start the necessary observing programs including the apecs CLI. The latter can also be simply started by typing apecs in a terminal window.
Please configure the "jlog" logging client to display only messages from the Observing Engine (use View/Filters/Add/File name/Wildcard "apexObs*") and set the cache to 5000 messages. This configuration can unfortunately not yet be done automatically.
It is recommended to use two different computers (e.g. tacora and pular) to start the two different scripts so that you can view all information simultaneously. Please choose a computer with sound card and speakers to run the observing clients so that you do get the audio notifications at the end of the scans.
Please consult table 4.1 if you need to start some of the monitoring or observing applications individually.
If several groups in different locations need to see the same session, it is recommended to use xkibitz to replicate the apecs CLI session (see the xkibitz man page for instructions).
To observe with APEX one uses the "apecs" scripting language. "apecs" is based on "(I)Python" ([8]) and provides a number of special commands to set up several different standard observing modes. Normal Python programmatic structures and macro capabilities can be used from within "apecs". Note that IPython allows to write commands which begin directly at the prompt without the usual brackets "()" that Python requires. However, "apecs" macros that define user commands (see section 3.4) need to be written using the brackets.
The apecs commands lead to the creation of a so called "Scan Object" that is then sent to the "Observing Engine". The "Scan Object" contains all the setup information needed to control the instruments and the telescope. It is also being used for status displays and to fetch information needed for the MBFITS raw data files.
To process your data, please use the GILDAS software for spectral line data and BoA for bolometer data. These packages are available on all APECS computers.
At the end of the observing session please stop all clients using the stopAPECSMonClients and stopAPECSObsClients scripts before logging out.

3.2 apecs commands

The apecs commands are divided into several groups to handle source setup, instrument setup and pattern setup. In addition to that there are commands for various calibrations and some miscellaneous commands. Type "help(apecs)" at the "APECS>" prompt to get an overview and "help(command)" to get a description of the individual commands and their current default values. User parameters are usually taken as the next default except for a few cases where this does not make sense (e.g. "source", "pcorr", "fcorr", etc.).
APEX does have a 30 degree Sun avoidance zone. There is an automatic avoidance mechanism that moves the telescope around the Sun if necessary. This mechanism works only when using apecs commands.

Observers **must not** move the telescope with the `apexAntMount_special` GUI since it does **not** know about the Sun avoidance zone !

3.2.1 General

3.2.1.1 `project_id`

`project_id( '<ID>')`

Defines the current project ID and with it the naming of the MBFITS raw data and CLASS scientific data files. It is very important to set this correctly at the beginning of each session in order to be able to identify your data later on ! `apecs` asks for the ID when is is started.
The project ID is assigned by the program committees or by the APEX station manager. The project ID "O-[K]PP.C-NNNN[R]-YYYY" is structured as follows:

- O is the origin of the project:
  - M: MPIfR
  - E: ESO
  - O: OSO
  - G: Germany (Verbundforschung)
  - C: Chile
  - P: PI Project
  - X: External (e.g. Berkeley)
  - K: Key program
  - A: APEX (staff time)
  - T: Technical / Maintenance / Calibration
- K PP is the ESO proposal identifier (K=0,1,2 (optional; omit for non-ESO projects)) and period (PP) (may not apply to the other partners; use "00" in that case)
- C is the (ESO) category (A-F) (may not apply to the other partners; use "F" in that case):
  - A: COSMOLOGY
  - B: GALAXIES AND GALACTIC NUCLEI
  - C: INTERSTELLAR MEDIUM, STAR FORMATION and PLANETARY SYSTEMS
  - D: STELLAR EVOLUTION
  - E: UNDEFINED
  - F: UNDEFINED
- NNNN is a sequential number
- R is the ESO run ID letter (optional; omit for non-ESO projects)
- YYYY is the year

3.2.1.2 `operator_id`

`operator_id( '<operator initials>')`

Set the operator ID for use in the MBFITS raw data file and in the logs.

3.2.1.3 `observer_id`

`observer_id( '<observer initials>')`

Set the observer ID for use in the MBFITS raw data file and in the logs.

3.2.1.4 show

```
show( )
show( <Scan object>)
```

Show the current observing mode setup. Optionally, one can pass a user defined scan object to show its setup.

3.2.1.5 go

```
go( )
```

Move the telescope to the desired center position and set up the instruments. Usually used to prepare the instrument setup for manual receiver tuning.

3.2.1.6 cancel

```
cancel( )
```

Cancel the ongoing scan.

3.2.1.7 save_history

```
save_history( '<File name>')
```

Save the `apecs` command history into a file. The file may be edited and used as a macro later on. Macros are executed using the `run_macro` command.

3.2.1.8 run_macro

```
run_macro( '<File name>')
```

Run an `apecs` command macro. The file may contain IPython and standard Python commands. Note that the definition of user commands using the Python `def` constructor need to be loaded with the Python `execfile` command due to namespace issues. In that case, one must type all brackets in the macro (see also section 3.4).

3.2.1.9 save_defaults

```
save_defaults( '<File name>')
```

Not yet implemented. Will save the current user command defaults to a file.

3.2.1.10 load_defaults

```
load_defaults( '<File name>')
```

Not yet implemented. Will load user command defaults from a file.

3.2.1.11 reset_defaults

```
reset_defaults( )
```

Not yet implemented. Will load system defaults for all commands.

3.2.1.12 observe

```
observe( )
observe( <Scan object>)
```

The internal method to submit a scan to the "Observing Engine". May be used to submit non-standard "Scan Objects".

3.2.1.13 continuous_data

```
continuous_data( 'on' | 'off')
```

Switch between subscan based ('off') or continuous ('on') data taking during the whole pattern, i.e. also when turning around to the next row or column, etc. Turning this mode on is recommended only for bolometer array observations. For all other observing setups, the online and offline data calibration and reduction would no longer work.

3.2.1.14 skip_hardware_setup

```
skip_hardware_setup( 'on' | 'off')
```

Select whether or not hardware should be set up for a scan. If the flag is set, then hardware is not configured if the setup is identical to the one used in the preceeding scan and if that scan has been completed less than 30 minutes ago. This saves scan overhead times, but it is somewhat risky as the correct setup is no longer guaranteed. One needs to take care that nobody modifies hardware parameters manually. The "go" command ignores this flag and always enforces a hardware setup.

3.2.2 Catalogs

Define user source and line catalogs.

3.2.2.1 sourcecats

```
sourcecats( '<File name>')
sourcecats( ['<File name 1>',
            '<File name 2>',
            ...])
```

Define the paths to user source catalog files in IRAM PdB format to specify source coordinates (equatorial J2000 or horizontal; no other systems are currently supported) and radial source velocity (LSR). Table 3.1 shows an example of a source catalog. The entries must be stored in a simple text file in the given format.

Source name(s)	System	Epoch	Lambda (RA or Az)	Beta (Dec or El)	Velocity frame	Velocity
Orion|BN-KL	EQ	2000.0	05:35:14.16	-05:22:21.5	LSR	8.0
Stow|Park	HO		180:00:00.00	15:00:00.0		

Table 3.1: Example of an APEX source catalog.

3.2.2.2 ephemerides

```
ephemerides( '<File name>')
ephemerides( ['<File name 1>',
              '<File name 2>',
              ...])
```

Define the paths to user ephemeris files in xephem "edb" format specifying orbital elements of solar system objects.

3.2.2.3 linecats

```
linecats( '<File name>')
linecats( ['<File name 1>',
           '<File name 2>',
           ...])
```

Define the paths to user line catalog files to specify transition details. Table 3.2 shows an example of a line catalog. The entries must be stored in a simple text file in the given format.

Transition name	Frequency	Unit	Sideband
CO(4-3)	461.040750	GHz	LSB
CO(3-2)	345.795969	GHz	USB

Table 3.2: Example of an APEX line catalog.

3.2.3 Instruments

This group of commands is used to define the instrument(s) to be used for the next scan. Instruments are composed of a frontend and a backend, thus they are also called "frontend-backend combinations" or FEBEs. There are bolometer (continuum) frontends and heterodyne frontends. Bolometers can be connected to continuum backends only. Heterodyne frontends can be connected to continuum backends (typically used for calibration, pointing, focus and skydip) and to spectral line backends (for single point integrations or maps).
Table 3.3 gives an overview of the current frontends and the ones expected until the end of 2006. Table 3.4 shows the backends and their specifications. The CLASS acronyms in both tables are being used to create the CLASS header variable `TELESCOPE` according to the naming scheme `AP-<2-letter frontend acronym><2-digit pixel number>-<2-letter backend acronym><2-digit baseband number>`. Finally, tables 3.5 and 3.6 show the possible FEBE combinations.

3.2.3.1 frontends

```
frontends( '<Frontend name>')
frontends( ['<Frontend name 1>',
            '<Frontend name 2>',
            ...])
```

Select the frontends to be used for the next scan.

Frontend	CLASS Acronym	Type	Receiver cabin	Number of pixels	Tuning Range [GHz]	Bandwidth [GHz]	Status
HET230	H2	Heterodyne (2SB)	A	1	211–275	4	Facility
HET345	H3	Heterodyne (DSB)	A	1	279–381	4	Facility
HET460	H4	Heterodyne (2SB)	A	1	385–500	4	Facility
FLASH460	F4	Heterodyne (DSB)	A	1	420–500	2	PI
FLASH810	F8	Heterodyne (DSB)	A	1	780–880	4	PI
CONDOR	CR	Heterodyne (DSB)	A	1	1370–1500	0.7	PI
CHAMP690	C6	Heterodyne (SSB)	B	7	600–720	4	PI
CHAMP810	C8	Heterodyne (SSB)	B	7	790–950	4	PI
LABOCA	LB	Bolometer	C	295	345	˜60	Facility
BOLO350	B3	Bolometer	C	37	810	N/A	Facility
BOLOSZ	SZ	Bolometer	C	331	150	˜50 (TBC)	PI
ARTEMIS450	A4	Bolometer	A	256	666	N/A	PI
ARTEMIS200	A2	Bolometer	A	256	1499	N/A	PI
HOLO	HO	Holography	C	1	92.4	0.56	Techn.

Table 3.3: APEX heterodyne and continuum frontends. The receiver cabins are named A (left Nasmyth), B (right Nasmyth) and C (Cassegrain).

Backend	CLASS Acronym	Type	Bandwidths per input [MHz]	Number of inputs	Numbers of spectral channels	Status
FFTS1	F1	Spectral	1000	2	16384, 8192, ... 4, 2, 1	Facility
FFTS2	F2	Spectral	1000	2	16384, 8192, ... 4, 2, 1	PI
MACS	MA	Spectral	1000, 500, 250	32	2048, 1024, 512	PI
PBE_A	PA	Continuum	Frontend bandwidth	8	1	Facility
PBE_B	PB	Continuum	Frontend bandwidth	8	1	Facility
PBE_C	PC	Continuum	Frontend bandwidth	8	1	PI
PBE_D	PD	Continuum	Frontend bandwidth	8	1	PI
ABBA	AB	Continuum	Frontend bandwidth	320	1	Facility
SZACBE	ZA	Continuum	Frontend bandwidth	331	1	PI
SZDCBE	ZD	Continuum	Frontend bandwidth	331	1	PI
ARTBE	AR	Continuum	Frontend bandwidth	256	1	PI

Table 3.4: APEX spectral and continuum backends

3.2.3.2 <Frontend name>.feeds

```
<frontend name>.feeds( ref='default' | <Reference pixel number>,
                       select='all' | [<Pixel number list >] | 'circle<radius>' |
                              '<Specific geometry keyword>')
```

Configure the frontend feed setup. The reference feed number defined by "ref" shifts the receiver pointing model to center the observations on the new feed and tells the online display software which data to show. One needs to provide a number or the string "default" to go back to the original value.
The "select" parameter allows to restrict the number of feeds to be used. One can either specify an explicit list of feeds (e.g. [1,2,4,6]) or use specific geometric selections like "wedge1", "wedge2", etc. or "circle¡radius¿" (all pixels within the radius given in arcsec from the coordinate system center).
Currently, the following specific geometries are defined:

– BOLOSZ: 'wedge1', 'wedge2', ..., 'wedge6' (selects the corresponding wedge of pixels).

	FFTS1	FFTS2 (MPIfR only)	MACS	PBE_A	PBE_B	PBE_C	PBE_D
HET230	X	–	–	X	–	–	–
HET345	X	–	–	X	–	–	–
HET460	X	–	–	X	–	–	–
FLASH460	X	X	–	X	–	–	–
FLASH810	X	X	–	X	–	–	–
CONDOR	X	X	–	X	–	–	–
CHAMP690	X	X	X	–	–	X	–
CHAMP810	X	X	X	–	–	–	X
HOLO	–	–	–	–	X	–	–

Table 3.5: Possible FEBE combinations for the APEX heterodyne frontends.

	ABBA	SZACBE	SZDCBE	ARTBE
LABOCA	X	–	–	–
BOLO350	X	–	–	–
BOLOSZ	–	X	X	–
ARTEMIS450	–	–	–	X
ARTEMIS200	–	–	–	X

Table 3.6: Possible FEBE combinations for the APEX bolometer frontends.

3.2.3.3 <Continuum frontend name>

```
<Continuum frontend name>( gain=<Amplifier gain>)
```

Configure the continuum frontend. The gain parameter is used by some frontends. Typical values are 1, 2, 4, 8, etc. The parameter is only used if there is an amplifier CORBA object present in the system.

3.2.3.4 <Heterodyne frontend name>

```
<Heterodyne frontend name>( mode='dsb' | 'ssb',
                            ratios=[<Image to signal band gains for feed 1>,
                            <... for feed 2>, ..., <... for feed N>],
                            fthrow=<Frequency switching throw> |
                            (<Phase 1 throw>, <Phase 2 throw>),
                            harmonic=<Number>)
```

Configure basic heterodyne frontend parameters. The mode can be DSB or SSB. The ratios are the linear image to signal band gain ratios for all feeds in numerical order. The frequency switching throw(s) are given in MHz. Usually, symmetric switching (+/- fthrow) is assumed. Asymmetric switching can be commanded using a list of throws (e.g. (-10.0,15.0)). The offsets are being used for the frequency switching observing mode which is activated by the `fsw` command. Note that the Observing Engine automatically chooses the recommended harmonic number if the user sets this to zero, which is the default. A non-zero number will override the system defaults.

3.2.3.5 <Heterodyne frontend name>.line

```
<Heterodyne frontend name>.line( name='<Line name>',
                                 frequency=<Number>,
                                 sideband='lsb' | 'usb',
                                 unit='GHz')
```

Select the line to be used for the heterodyne frontend tuning. The system will try to find the line in the catalogs if only the name is given.

3.2.3.6 <Heterodyne frontend name>.derotate

```
<Heterodyne frontend name>.derotate( mode='ca' | 'ho' | 'eq',
                                     angle='full' | 'half' | <User angle in degrees>)
```

Define derotation setup of array receivers. The angle in degrees defines the rotation of the array relative to the selected coordinate system. This is used to select the spatial sampling setup. Alternatively, one can specify 'full' or 'half' to automatically select those sampling setups. The mode can be 'CA' (fixed, no derotation), 'HO' (horizontal derotation) or 'EQ' (equatorial derotation).

3.2.3.7 <Frontend name>.backends

```
<Frontend name>.backends( '<Backend name>)
<Frontend name>.backends( ['<Backend name 1>',
                           '<Backend name 2>',
                           ...])
```

Connect continuum and/or spectral line backends to the selected frontends. Note that `apecs` automatically assigns backend inputs. In case of continuum backends, the wiring is not computer configurable and the inputs are selected according to the hardware setup stored in `apecs`. In case of spectral line backends, some connections are computer configurable (e.g. via the Common IF). `apecs` administers those resources automatically and assigns a number of spectral line backend inputs according to the number of receiver feeds.
In both cases, the selected inputs are named a "backend section group". The `<Frontend name>.backends` will print a group number which is needed later on when configuring the properties of a particular "backend section group".

3.2.3.8 <Continuum backend name>

```
<Continuum backend name>( dumptime=<Time in seconds>, gain=<Gain (1, ..., 100)>)
```

Configure the continuum backend dump time and gain. The default dump time is 0.0 s which means that the system determines the time automatically. The dump time is given in seconds and can range from 0.0 to 4.0 s. The gain can be an integer between 1 and 100. It controls the dynamic range by scaling the input signal by 1/gain.

3.2.3.9 <Spectral backend name>

```
<Spectral backend name>( dumptime=<Time in seconds>)
```

Configure the spectral backend dump time. The default is 0.0 s which means that the system determines the time automatically. The dump time is given in seconds and can range from 0.0 to 4.0 s.

3.2.3.10 <Spectral backend name>_group

```
<Spectral backend name>_group( group=<Group number>,
                               bandwidth=<Bandwidth in MHz>,
                               numchan=<Number of channels>,
                               offset=<Offset in MHz>,
                               linename='<Alternative line name>',
                               sections=[<List of backend sections>])
```

Configure a spectral line backend section group. The necessary group number is being displayed by `apecs` when connecting the backends to the frontends. It can also be queried using the `show()` command.
Bandwidth and section group radio frequency offset are given in MHz. The numbers of channels can be chosen among the possible values for the particular backend. Optionally, a line name can be given if the section group is centered on a different line than the main one defined for the frontend. Leave the line name empty for composite spectra so that one can simply add the spectra in CLASS. Finally, the default sections making up this group can be overridden if needed. They must be given as a list.

3.2.4 Target

Commands in this section are used to define the target coordinates.

3.2.4.1 source

```
source( '<Name>')
source( name='<Name>',
        x=('<x-Coordinate>', <Unit>),
        y=('<y-Coordinate>', <Unit>),
        system='eq' | 'ho',
        epoch=2000.0,
        velocity=<Velocity in km/s>,
        frame='LSR' | 'HELIO')
```

The source command can be used to define a center position in either horizontal or equatorial (J2000 only at this point) coordinates. The parameters may be specified manually or can be read from a catalog (see `sourcecats`). The major solar system objects are automatically recognized by their names. Ephemeris files for a number of other solar system objects (comets, etc.) are installed in the system. User ephemeris files in `xephem`'s "`edb`" format can be introduced into the system (ask one of the software engineers to copy them to right place before the observations begin).
Note that the `TICS` software always uses descriptive coordinates, i.e. the new coordinate system centered at the source is using great circles in the sky. This is different compared to many other telescope control systems but it makes sense for the observing, in particular for wobbler observations where the wobbler motion occurs in that coordinate system too.

3.2.5 Calibration

3.2.5.1 calibrate

```
calibrate( mode='cold | hot',
           time=<Time per point in seconds>,
           autolevel='on' | 'off')
```

Perform a calibration scan. Default is a sequence of SKY-HOT-COLD. If the mode is set to 'HOT', only SKY-HOT is being observed. Auto-levelling should usually be turned on. For some instruments levelling takes a very long time. This step can be skipped by specifying autolevel='off'.

3.2.5.2 skydip

```
skydip( azimuth='current' | <Angle in degrees>,
        am_stop=<Number>,
        am_start=<Number>,
        points=<Number>,
        time=<Time per point in seconds>)
```

Perform a skydip scan at the given azimuth (default is the current value) between the two air masses defined in the command. Do not specify an air mass of exactly 1.0 to avoid problems with the antenna control system.

3.2.5.3 point

```
point( length=<Number>,
       unit='arcsec',
       time=<Time per subscan in seconds>,
       mode='otf' | 'ras',
       points=<Number>,
       direction='x' | 'y')
```

Perform a cross scan in OTF or raster mode to check the local pointing near the source.

3.2.5.4 pcorr

```
pcorr( )
pcorr( <Azimuth offset in arcsec> | '*' | 'f',
       <Elevation offset in arcsec> | '*' | 'f')
```

Correct the receiver pointing model. Usually the numbers are fetched automatically (≡ 'f') from the online calibrator. For line pointings one currently needs to run the reduction script (`lpoint.class`) manually offline and enter the numbers manually as displayed by the script. Specifying '*' keeps the previous value.

3.2.5.5 pcorr_reset

```
pcorr_reset( )
```

Reset the pointing model corrections to zero. Note that this command currently works globally, i.e. independently of which frontend is in use.

3.2.5.6 focus

```
focus( amplitude=<Amplitude in mm or arcsec>,
       time=<Time per subscan>,
       points=<Number of points>,
       axis='z' | 'y' | 'x' | 'xtilt' | 'ytilt',
       mode='pos' | 'neg' | 'sym')
```

Perform a focus scan to check the local focus point at the current elevation. The focus of the selected axis is moved in the range of `-amplitude` to `+amplitude` with the given number steps in between. The mode can be "pos"(itive) for increasing offsets, "neg"(ative) for decreasing offsets or "sym"(metric) for both, starting with increasing offsets. A measurement is taken for each focus setting and the online calibrator tries to fit a parabola to the data.

3.2.5.7 fcorr

```
fcorr( )
fcorr( delta=<Offset in mm or arcsec> | '*' | 'f',
       axis='z' | 'y' | 'x' | 'xtilt' | 'ytilt',
       febe=<FEBE to use for the fit>,
       group=<Section group number>)
```

Correct the local focus setting for a given axis. The number and the axis are fetched automatically (≡ 'f') from the online calibrator if no parameters are given to `fcorr`. Specifying '*' keeps the previous value.

3.2.5.8 fcorr_reset

```
fcorr_reset( )
```

Reset the pointing model corrections due to lateral hexapod shifts to zero. Note that this command currently works globally, i.e. independently of which frontend is in use. Note also that the receiver hexapod shifts are currently not set to zero by this command.

3.2.5.9 set_tilts

```
set_tilts( AN=<AN pointing model term>, AW=<AW pointing model term>)
```

Set tilt values in arcsec to override the current pointing model (base plus frontend) values completely. "reset_tilts" reverts back to the models. This command may be needed if the telescope tilts change on a short time scale of a few hours. Please ask the APEX staff about the `apecs` script to determine the new numbers.

3.2.5.10 reset_tilts

```
reset_tilts( )
```

Revert back to the tilts defined by the pointing models.

3.2.5.11 use_linear_sensors

```
use_linear_sensors( )
```

Define whether to use the linear sensor pointing correction. Accepts values 'on' and 'off'.

3.2.6 Observing Patterns

Observing patterns are set up using the `TICS` strokes. There are linear and arc strokes. Most of the APECS patterns are composed of linear strokes. The `spiral_setup` command uses an arc stroke.
Note that `TICS` sets up a new descriptive coordinate system centered on the source coordinates and that offsets are always applied along the great circles. This means that all patterns will look slightly different when being projected to encoder coordinates (azimuth / elevation) or equatorial coordinates (right ascension / declination).

3.2.6.1 offset

```
offset( x=<x-Offset>,
        y=<y-Offset>,
        unit='arcsec' | 'arcmin',
        mode='rel' | 'abs',
        system='eq' | 'ho',
        epoch=2000.0)
```

Define an offset position for the next pattern to be centered on. Note that observing a horizontal pattern on an equatorial offset position leads to shifting the user coordinate system to that offset position before applying the horizontal offsets.

3.2.6.2 reference

```
reference( x=<x-Offset>,
           y=<y-Offset>,
           time=<Time in seconds>,
           on2off=<Number of On's per Off>,
           unit='arcsec' | 'arcmin',
           mode='rel' | 'abs',
           system='eq' | 'ho',
           epoch=2000.0)
```

Define a sky reference position for any On-Off type observation including calibrations. The ideal time on the reference position is calculated automatically for all observing modes unless a non-zero value is defined here. The reference positions are always taken before the target measurements.

3.2.6.3 use_ref

```
use_ref( 'on' | 'off')
```

Select whether to use the reference position in an observing mode. Note that the system automatically skips the reference if only continuum backends are connected. Thus it is usually not necessary to change the default of 'on' for `use_ref`.

3.2.6.4 on

```
on( time=<Time per point>)
```

Observe a single point centered on the offset defined by the `offset` command. Uses the reference if `use_ref` is "1" and if spectral backends are being used.
Note that the resulting `CLASS` data are projected into encoder coordinates (horizontal system) or J2000 coordinates (equatorial system). Due to the use of descriptive coordinate systems in APECS, the projected values are not the same as the commanded descriptive values.

3.2.6.5 raster

```
raster( xlen=<x-Length>,
        xstep=<x-Step>,
        ylen=<y-Length>,
        ystep=<y-Step>,
        time=<Time per raster point>,
        direction='x' | 'y',
        zigzag=1 | 0,
        unit='arcsec' | 'arcmin',
        system='eq' | 'ho',
        epoch=2000.0,
        mode='ordered' | 'jiggle')
```

Set up a rectangular raster pattern given by sizes and steps in both directions. The time is used per raster point. If zigzag==1, the raster is done bi-directionally, reducing the telescope overhead. The direction defines the fastest varying axis for an ordered setup. The mode can be "ordered" to obtain a defined sequence of points or "jiggle" to observe the raster points in random order. Reference subscans are inserted according to the settings given in the "reference" command except for wobbler mode.
Note that the resulting `CLASS` data are projected into encoder coordinates (horizontal system) or J2000 coordinates (equatorial system). Due to the use of descriptive coordinate systems in APECS, the projections are no longer simple rectangles.

3.2.6.6 hexa

```
hexa( time=<Time per raster point>,
      sampling='full' | 'half' | 'extend',
      beamsize=0.0 | <Beam size>,
      unit='arcsec' | 'arcmin',
      feedsep=<Feed separation in number of beam sizes>,
      system='eq' | 'ho',
      epoch=2000.0,
      mode='ordered' | 'jiggle')
```

Set up a hexagonal raster pattern to map out the footprint area of a (hexagonal) array receiver (e.g. CHAMP+). The time is used per raster point.
The sampling can be "full", "half" or "extend" for fully / half sampled maps or to extend a half sampled map with the remaining points to obtain a fully sampled one. The beam size is automatically calculated from the highest sky frequency for the current scan if a value of 0.0 is given. Otherwise the non-zero user value is used.
The feed separation is the radial distance of neighboring feeds in number of beam sizes (for CHAMP+ this number is 2). The mode can be "ordered" to obtain a defined sequence of points or "jiggle" to observe the raster points in random order. Reference subscans are inserted according to the settings given in the "reference" command except for wobbler mode.
Note that the array derotation angle should be 0.0 in "eq" mode.

3.2.6.7 otf

```
otf( xlen=<x-Length>,
     xstep=<x-Step>,
     ylen=<y-Length>,
     ystep=<y-Step>,
     time=<Time per raster point>,
     direction='x' | 'y',
     zigzag=1 | 0,
     unit='arcsec' | 'arcmin',
     system='eq' | 'ho',
     epoch=2000.0,
     mode='ordered' | 'jiggle')
```

Perform a rectangular "On-The-Fly" mapping. The telescope scans along map rows or columns with a given speed while data is being written once per defined time interval.
If zigzag==1, the OTF map is done bi-directionally, reducing the telescope overhead. The direction defines the scanning axis. The angle defines the position angle of the map relative to the coordinate system. The mode can be "ordered" to obtain a defined sequence of points or "jiggle" to observe the OTF lines in random order. Reference subscans are inserted according to the settings given in the "reference" command except for wobbler mode.
Note that the steps along the scanning axis should be ideally chosen to be 0.45 times the current beam size. The beam will become elongated if the steps are chosen to be too large.
Note that the resulting `CLASS` data are projected into encoder coordinates (horizontal system) or J2000 coordinates (equatorial system). Due to the use of descriptive coordinate systems in APECS, the projections are no longer simple rectangles.

3.2.6.8 drift

```
drift( azsize=<map size in azimuth>,
       elsize=<map size in elevation>,
       elstep=<step size in elevation>,
       lineartime=<time for linear stroke>,
       turntime=<time for turn-around>,
```

```
mode='OPEN' | 'CLOSED',
azunit='deg' | 'arcmin' | 'arcsec',
elunit='arcmin' | 'deg' | 'arcsec')
```

Set up a rectangular azimuthal On-The-Fly drift pattern along lines of constant elevation with half circle turn-arounds to minimize microphonics through telescope accelerations. The pattern is centered on a horizontal source or around the average azimuth and elevation of an equatorial source. The source thus drifts through the line of sight during the scan. Note that the source name is changed to <Original name>-DR_<Az>_<El> to show that a fixed position in horizontal coordinates was observed.
The extent of the pattern is given by the size of the linear portion of the azimuth stroke and a size in elevation. In "open" mode, rows spaced by the elevation step size are set up with zig-zag scanning. In "closed" mode, a "race-track" like closed loop pattern is set up. In this mode the elevation size can be zero to obtain a simple sweep pattern at one elevation value. Repetitions of these patterns are controlled with the "repeat" command. In case of the "race-track" pattern, all cycles are performed in one subscan. The durations (in seconds) of the linear stroke and the turn-around can be set individually, however the turn-around time must be at least 2 seconds to avoid extreme telescope accelerations.
The default backend dump time for drift scans is 0.5 s. This can be overridden by setting individual backend dump times using the backend setup commands.
This mode is mainly used for the SZ project to minimize signal modulations by the earth's atmosphere.

3.2.6.9 spiral_setup

```
spiral_setup( r0=<Starting radius>, r_dot=<Radial veolcity>,
              phi_dot=<Angular velocity>,
              r0_unit='arcsec' | 'arcmin' | 'deg',
              r_dot_unit='arcsec/s' | 'arcmin/s',
              phi_dot_unit='deg/s' | 'arcmin/s')
```

Set up a spiral On-The-Fly pattern for use with other observing commands. The spiral is performed relative to the current offset position. It is defined by a start radius, a radial velocity and an angular velocity. The default backend dumptime 0.5 s. This can be overriden with the backend setup commands. The total time for the spiral is defined by the individual observing commands like "on", "raster", "point", etc.

3.2.6.10 use_spiral

```
use_spiral( 'on' | 'off')
```

Define whether to use spiral strokes set up by the spiral_setup command. Accepts values 'on' and 'off'. When enabled, any normal tracking stroke will be replaced with a spiral. This mode is mainly meant for bolometer observations.

3.2.6.11 repeat

```
repeat( <Number of pattern repeats>)
```

Set the number repeats for any given observing mode. Used to repeat a given pattern (e.g. a raster) <N> times. Note that `repeat` also applies to the pointing and focus commands. Selecting a new source or a new backend setup will reset the repeat count to 1.

3.2.7 Switch Mode

3.2.7.1 tp

```
tp( )
```

Select total power mode. All subsequenct scans are performed without wobbler or frequency switching.

3.2.7.2 wob

```
wob( amplitude=<Wobbler amplitude in arcsec>, rate=<Wobbler rate in Hz>,
     mode='sym' | 'neg' | 'pos', blank=-1 | <Blank time in ms>)
```

Select wobbler mode. All subsequent scans are performed using the wobbler with the parameters given here. The wobbler amplitude is given in arcsec, the wobbling rate in Hz. The mode can be "sym"(metric), "neg"(ative) or "pos"(itive) to select the reference point location. The optional blank(ing) parameter allows to override the system setting of the blank time to be used for wobbler observations. The time is given in milli-seconds. A value of -1 selects the automatic mode.

3.2.7.3 fsw

```
fsw( rate=<Frequency switching rate in Hz>, blank=-1 | <Blank time in ms>)
```

Select frequency switching mode. All subsequent scans are performed using frequency switching. The frequency switching rate is given in Hz. The frequency throws are defined individually for each frontend using the ¡heterodyne frontend name¿ commands. The optional blank(ing) parameter allows to override the system setting of the blank time to be used for frequency switching observations. The time is given in milli-seconds. A value of -1 selects the automatic mode.

3.2.8 Antenna

3.2.8.1 tolerance

```
tolerance( <Tolerance radius in arcsec>)
```

Define the required initial tracking accuracy. A subscan will only begin if the telescope is within the given radius for some time ticks of 48ms.

3.2.8.2 park

```
park( )
```

Move the telescope to the stow position (South at 15 degrees elevation) and switch to SHUTDOWN mode.

3.2.8.3 stow

```
stow( )
```

Move the telescope to the stow position (South at 15 degrees elevation), insert the stow pins and switch to SHUTDOWN mode. Note that the Observing Engine will refuse to observe any further scan until the telescope is un-stowed.

3.2.8.4 unstow

```
unstow( )
```

Retract the stow pins and park the telescope at the stow position (South at 15 degrees elevation). The telescope is now prepared for observations.

3.3 A typical Observing Session

The following sections show some example sequences of `apecs` commands to set up source and instruments and to perform some typical continuum and spectral line observations.

3.3.1 Source Setup

```
# Define a source catalog
sourcecats('user.cat')
# Define a line catalog
linecats('user.lin')
# Load a source from the catalog
source('bn-kl')
```

3.3.2 Continuum Instrument Setup

```
# Define frontend(s)
frontends('laboca')
laboca gain=2
# Continuum backends
laboca.backends('abba')
# Select continuum backend gain
abba(gain=4)
```

3.3.3 Spectral Line Instrument Setup

```
# Define frontend(s)
frontends(['flash460', 'flash810'])
# Define lines
flash460.line('CO(4-3)')
flash810.line('CO(7-6)')
# Spectral backend setup.  Note the group numbers that the CLI displays.
# They can also always be queried using the show() command.
flash460.backends('ffts1')
flash810.backends('ffts1')
# Configure the spectral backends
ffts1_group(1, numchan=8192)
ffts1_group(2, numchan=2048)
```

3.3.4 Initial Calibrations

```
# Select a submm calibrator
source('mars')
# Select continuum backend(s) also for heterodyne receivers.
flash460.backends('pbe_a')
flash810.backends('pbe_a')
# Perform a SKY-HOT-COLD measurement
calibrate()
# Perform an OTF cross scan.  The online calibrator automatically fits a
# Gaussian and provides the offsets to correct the pointing model.
point()
# Fetch the pointing offsets and apply them to the receiver pointing model.
pcorr()
# Perform a focus measurement (default "Z", i.e. radial focus)
focus()
# Fetch the focus offsets and apply them to the receiver focus position
# and auto-correct its pointing model for lateral focus shifts.
```

```
fcorr()
```

3.3.5 Continuum Observations

```
# Select frontends and continuum backends as shown above
# Select a science target
source('orion')
# Define a reference position for the calibration.
reference(-3600, 0)
# Perform a SKY-HOT-COLD measurement
calibrate()
# Perform an OTF map.  Note that the reference is not being used here
# since no spectral backend is connected.
otf(3600, 1, 200, 8, time=0.005, system='eq')
```

3.3.6 Spectral Line Observations

```
# Select frontends and spectral line backends as shown above
# Select a science target
source('orion')
# Define a reference position.  Note that the time will be automatically
# computed for the different observing modes if it is set to 0.0 in the
# reference command (this is the default).
reference(-3600, 0)
# Perform a SKY-HOT-COLD measurement
calibrate()
# Perform an OTF map.  Note that the reference is being used automatically
# if spectral backends are connected.
otf(400, 3, 30, 3, time=0.1, system='eq')
```

3.4 Macros and Loops

Macros are supported by the commands `run_macro` (IPython syntax without brackets) and `execfile` (standard Python syntax with brackets). To run a set of `apecs` commands, edit them into a text file and execute them from the `APECS>` prompt by typing `run_macro '<filename>'` or `execfile '<filename>'`.
Often one wants to repeat a certain set of commands or macros in a loop. Use the Python programmatic structures to accomplish this. Note that Python is strict about keeping the same amount of indentation for a given loop or branch level.
One can also use Python variables to construct more complicated complex patterns alternating between calibrations and target observations. One example of using loops and variables would be the following set of OTF maps:

```
reference(-1800, 0, on2off=1, mode='rel', system='EQ')
# Loop between -60 and 120 in steps of 60.
# Note that xrange works with integer numbers.  Floating point numbers are
# provided by the Numeric.arange() function.
for yoff in xrange(-60, 121, 60):
    offset(20, yoff)
    calibrate()
    otf(270, 5, 45, 5, time=1, direction='x')
```

3.5 User defined Commands

More complex user commands can be defined in `apecs` via Python functions. For example, the switch to continuum backends and a subsequent pointing can be grouped together to define a new user command.

The following example defines a special pointing command for FLASH observations. The two parameters `length` and `time` are defaulted to some values but can be overwritten like for normal `apecs` commands. Please consult the Python documentation at `www.python.org` for more information on programming. The command definition must be read into `apecs` using `execfile` since `run_macro` would load it into another namespace unavailable to the interactive interpreter.

```
def flash_point(length=180,time=30):
    frontends(['flash460','flash810'])
    flash460.backends('pbe_a')
    flash810.backends('pbe_a')
    point(length=length,time=time,mode='otf')
```

The newly defined commands can be used like any other `apecs` command.

3.6 Notes and Caveats

- Note that scans are submitted to a queue which is 2-deep, i.e. while the current scan is being executed, the next one will be kept on hold within the `apecs` CLI giving the user a chance to cancel the submission by typing <CTRL-C> (be careful though about <CTRL-C> at the normal prompt (see below)).
- Do not type <CTRL-C> at the normal `APECS>` prompt. Due to a problem with the CORBA Python modules in ACS 2.0.1, this will cause a segmentation fault and end the "apecs" session. <CTRL-C> can only be used to cancel a scan submission that is on hold (see above). Once we port APECS to ACS 5.*, this will no longer be a problem.
- Do not type `help(<command>())` (note the second pair of brackets) as this will execute the command rather than print help on it. In particular, one may inadvertently submit a new scan to the Observing Engine. `help` must be used with just the command name, i.e. `help(<command>)`.
- Note that the help for some commands just prints some Python class information. We are working on a fix for that. Meanwhile you can get the help text by typing `help <command>.__call__`.
- Note that Python is strict about keeping the same amount of indentation for a given loop or branch level.
- Note that all APECS computers are running on TAI rather than UTC even though the time commands claim to display GMT (Linux does not have a TAI timezone). The local time is thus currently 33 seconds ahead of UTC. This number will increase by one whenever the next leap second will be introduced.
- For operators and astronomers on duty: Be **extremely** careful when using `apexAntMount_special` to move the telescope because it does not obey the Sun avoidance zone !

Chapter 4

Operating the APEX Telescope

4.1 Introduction

This section is aimed at APEX operators and (experienced) test observers. It describes how to start APECS and how to troubleshoot the system. The hints and recipes apply to the software status as of June 2006. The system is still evolving as development, debugging and improvements are still going on. Thus the content of this chapter will change with time and this document will be updated accordingly.
APECS is running on a number of control computers inside the antenna ("`instruments`", "`abm`"), in the control containers at Chajnantor (servers: "`control`", "`display`"; clients: `toco`, `leon`) and in the base at Sequitor ("`pular`", "`tacora`", "`apexdb2`, "`apexdev`", "`lastarria`"). All of those computers, except `abm`, are running under Linux (currently RedHat 7.2 due to using ACS 2.0.1). The `abm` computer is a VME system running VxWorks 5.4.
In addition to the APECS computers there are a number of embedded system computers ("`apexcool`", "`cs-a1`", "`ffts1`", `ffts2`, "`if_control`", and, to some extent, `instruments`) to control the instruments. Those computers run different versions of RedHat or SuSE Linux. Some future instruments (e.g. `ABBA`) will run under MS Windows. Embedded system computers are under the responsibility of the respective system developers. The interface to the embedded system is realized via ASCII SCPI commands sent over UDP sockets ([1]).
Fig. 4.1 shows the deployment of the APECS software on the various computers inside the telescope and in the control rooms at Chajnantor and in Sequitor.

4.2 Starting the APECS servers

The APECS servers provide the necessary infrastructure to perform observations. They are being started via a single script called `restartAPECSServers`. This script needs to run on the central APECS machine called `control`. It starts the core CORBA services and the APECS service applications such as Observing Engine, MBFITS Writer and Online Calibrator.
Currently, this is still to be done within a VNC server on `control` which one can start using the `vncs` alias. The VNC setup is needed in order to be able to view the display of the Online Calibrator also at other locations than the one where APECS was started. This restriction will be removed once we split up the Calibrator into a server and multiple result viewing clients. One needs to connect to the VNC server via typing `vncviewer -shared control:<display number (usually "1")>`.
It is, however, not recommended to use the VNC setup for normal observing. The CORBA middleware allows to run all clients on any of the APECS machines. The purpose is to distribute the work load among the machines. In the long run it is important not to overload the core computers with monitoring applications or status displays. Also the remote observing and/or monitoring will be performed using a VPN into the APECS system in order to be able to run applications locally at the remote institute. The system response times will thus be much shorter than with the current VNC setup. Please use `xkibitz` if simultaneous access to an `apecs` session is desired.
Before starting APECS, one should make sure that the machines are all idle and that the disks are not full. This is absolutely necessary since the startup script is currently time-critical. In any event one should

not run any other applications on the APECS server computers (`control`, `display` and `instruments`) anyways. Make especially sure that there is no web cam display running on any of the core computers at the site at Chajnantor as those use all the available CPU power.
For the server startup one needs to log on to `control` as user `apex` under which all operations are done. At the prompt on `control` type `restartAPECSServers -f` to start APECS.
There is also a simulation mode in which the telescope is being simulated and the real telescope is not moved. Everything else is identical to the real setup. This mode can be started by typing `restartAPECSServers -sf`. Note that the tracking in simulation mode is not as good as in the real hardware due to the non-aligned 10ms Linux scheduler. One thus needs to increase the tracking tolerance to about 5 arcseconds using the `tolerance` command in `apecs`.
After about 10 minutes the system will be up and running and a number of windows and displays will open. Tables 4.1 and 4.2 summarize the processes that need to be running for APECS to work. The troubleshooting section refers to the commands in this table when suggesting to restart part of the system.

4.3 Configurations

APECS reads a number of configurations from the area defined by the `$APEXCONFIG` environment variable:

- `$APEXCONFIG/Pointing`: Pointing models.
- `$APEXCONFIG/Catalogs`: System source and line catalogs.
- `$APEXCONFIG/Ephemerides`: Ephemeris files.

The pointing models are split into a master model (`MASTER.ptg`) which describes the overall antenna behavior. This is currently based on the optical pointing model and might be replaced later by bolometer radio pointing models.
The receiver delta pointing models are named after their `apecs` name, e.g. HET345.ptg. They contain only the deltas to be added to the master model. For Cassegrain receivers there are usually just two offset number, for Nasmyth receivers there are 4 constants.
The ephemeris files are stored in `xephem` "`edb`" format. Ephemerides for comets and distant objects are installed.
The scan number is being increased using the global scan number in `$APEXCONFIG/globalScanProperties`. The scan number should be reset to "1" on January 1st every year using the `initGlobalScanProperties` command.
The non-configurable instrument setup (i.e. wirings, etc.) is hardcoded in the `apexFEBEConfig.py` Python module which is located in the `$APEXROOT/lib/python/site-packages` directory. It is assumed that this stays stable once the instrument has been commissioned and accepted. Changing this setup is only possible under version control by a software responsible with authorization by the project manager or the station manager.
Note that the ethernet switch port used to connect the "ABM" VME computer needs to be configured to "100half" with "auto-negotiation" disabled. The MVME 761 board cannot handle "auto-negotiation". All other 100 Mbps copper wire ports should be configured to "100half" ("100full" may also working but has not yet been tested) with "auto-negotiation" enabled. The fiber ports must be set to "1000full".

4.4 Troubleshooting

While APECS is now much more stable than in the initial phases of the APEX commissioning, we still suffer from a number of problems. First and foremost check the messages in the "`jlog`" if an error occurs. Most conditions can be diagnosed by evaluating those messages. All logging messages are stored in the "`$APEXSYSLOGS`" directory. One will also find there all the details about the SCPI communication to the embedded systems.
For a proper debugging, one needs to have at least a coarse idea of how the system works. The details of the APECS design have been compiled in some training documents. Here we give just a short summary of the setup. The APECS system is using CORBA as the main communication mechanism. A number of CORBA services (naming, notification, CDB etc.) are started first. Without those services, the system will not work at all.

There is a central manager process that knows about the layout of the system, in particular which process is running on which machine. Next there are a number of so called container processes in which the CORBA objects are being instantiated. The CORBA objects are mostly used to map hardware functionality into the control system.
For APEX it was decided to not burden the instrument developers with CORBA software development which can be quite heavy. Instead we agreed on a simple text protocol based on the SCPI standard to send commands to the instrument control systems and to receiver monitor point information.
Thus the APEX CORBA objects are simply a wrapper between CORBA requests and corresponding SCPI commands sent via UDP socket connections to the embedded systems ([1]). They are therefore automatically generated from the IDL interface files. The ABM CORBA objects inherited from ALMA via the TICS software are different. They are implemented using the CAN bus communication directly without additional layers.
It is important to understand this layered structure of the APECS system. The top-level communication errors that the user sees can have different causes either on the CORBA container level or on the SCPI level. In ACS 2.0.1 it is impossible to route the nature of the error all the way up to the user. This may change in future versions of ACS. In the current setup one needs to examine all communication layers to debug the problems.
The collection of CORBA objects representing the hardware is being coordinated by the central "Observing Engine" process. It sets up all instruments, auxiliary devices and the antenna itself for each scan. If one of those setups fails, the whole scan is canceled. This is usually the point where observers turn to operators or software engineers for help. You then need to examine the system and the log messages to identify the offending sub-system.
Tables 4.3 to 4.4 summarize a number of known conditions and their usual solution. Sometimes the suggested measures might not resolve the problem and further debugging is needed. Always try to get the system back into operation by restarting only small portions of it rather than simply restarting everything. This way one learns more about the structure of the system and it is faster too.

4.5 Notes and Caveats

- The `restartAPECSServers` script runs under the `apex` account and tries to stop all APECS processes on all machines. However, `apex` is not (yet) able to stop observer processes running under the project account. It is thus possible that many old jobs accumulate over time. Thus please make sure that observing and monitoring clients are stopped at the end of each observing session using the `stopAPECSObsClients` and `stopAPECSMonClients` commands.
- Note that using only `stopAPECSServers` followed by `startAPECSServers` will **not** stop any clients. The idea is to leave the clients running so that they can re-connect once the servers are back up. This is not yet fully working under ACS 2.0.1 but may be fine for ACS 6 or higher. We currently recommend to always use `restartAPECSServers` or issue an explicit `stopAllAPECSClients` before stopping the servers.
- The `abm` computer's console is connected to `instruments` via an RS232 line which is accessible through the `minicom 1` command on `instruments`. One can thus access the `abm` independent of the ethernet to check its status or possibly reboot it by typing `reboot` at the `LABM>` prompt or, if it is not reacting anymore, by typing <CTRL-X>.
- **Never** type `exit` at the `abm` prompt as this closes the login shell and the computer needs to be rebooted. Use `logout` if you are connected via `rlogin` or `rsh`.

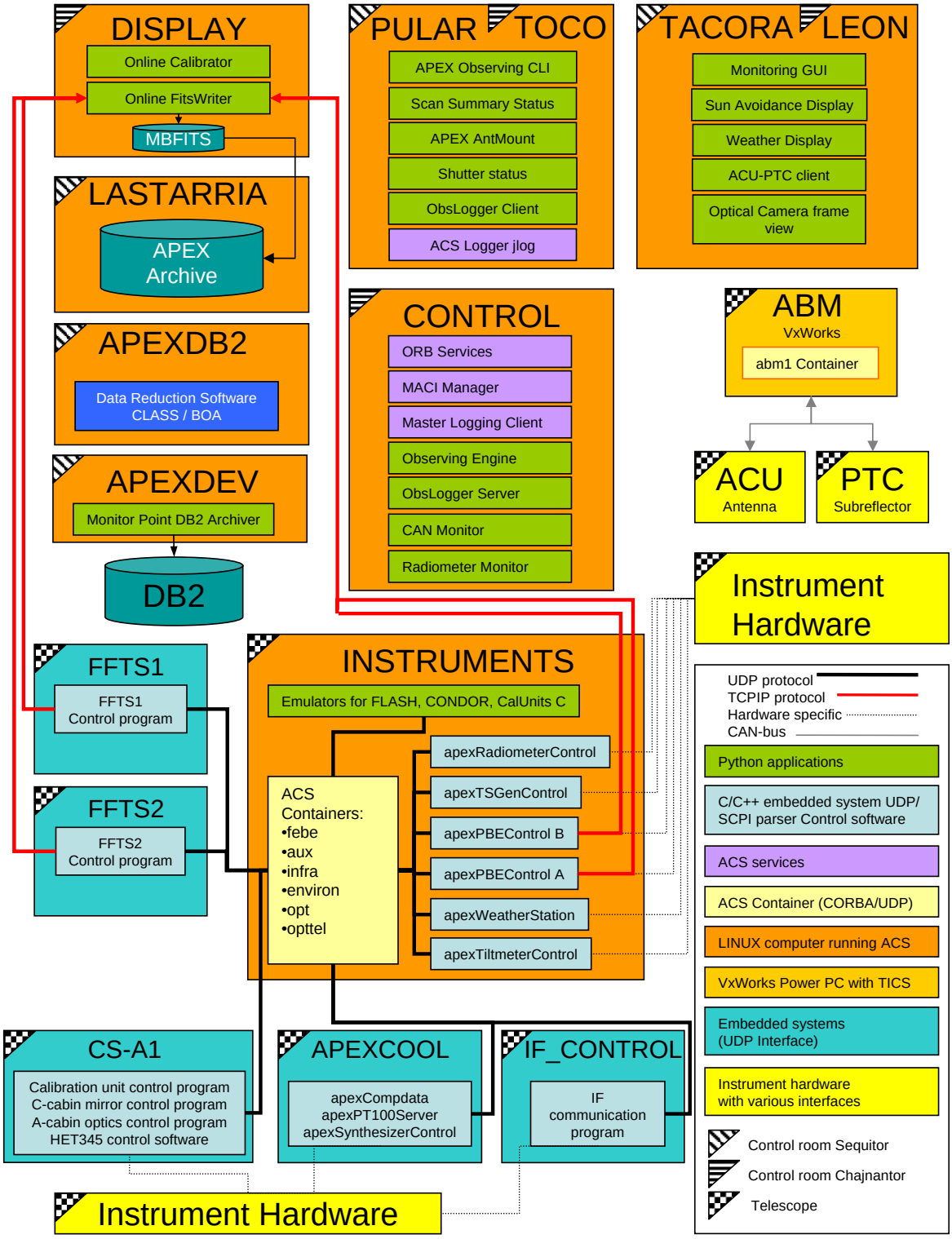

Figure 4.1: *APECS deployment at the site in Chile.*

Host	Applications	Commands
control	ORB Services (Naming, Notification, etc.)	`acsStartORBSRVC`
	Master logging client	
	MACI Manager	`maciManager`
	CAN Monitor	`canMonitor`
	Radiometer Monitor	`radiometerMonitor`
	Observing Engine	`apexObsEngine`
	Observation Logger Server	`apexObsLoggerServer`
instruments (APECS)	`febe` Container	`febeContainer start`
	`aux` Container	`auxContainer start`
	`infra` Container	`infraContainer start`
	`environ` Container	`environContainer start`
	`opt` Container	`optContainer start`
	`opttel` Container	`opttelContainer start`
instruments (Embedded System Servers included in APECS)	Weather station SCPI CID Server	`apexWeatherStation start`
	Radiometer SCPI CID Server	`apexRadiometer start`
	Tiltmeter SCPI CID Server	`apexTiltmeter start`
	TSGen SCPI CID Server	`apexTSGen start`
	PBE_A SCPI CID Server	`apexPBE_A start`
	PBE_B SCPI CID Server	`apexPBE_Bholo start`
	FLASH emulator	`emuEmbSys APEX_FLASH 0`
	SEST_RX emulator	`emuEmbSys APEX_SESTRX 0`
	CALUNITS:C emulator	`emuEmbSys APEX_CALUNITS_C 0`
display	Online MBFITS Writer	`apexOnlineFitsWriter`
	Online Calibrator	`apexOnlineCalibrator`
abm	`abm1` Container	`sp ActivateStart,"abm1"`
pular / toco	Observing CLI	`apecs`
	Scan summary status	`scanStatus`
	ACS logging messages	`jlog`
	Antenna position	`apexAntMount`
	Sun avoidance display	`plotAvoidance`
	Observation Logger Client	`apexObsLoggerClient`
	Shutter Control GUI	`shutterControl`
tacora / leon	Main monitoring client	`apexStatusDisplay \` `masterStatus.MasterStatus`
	Weather display	`apexWeatherDisplay`
	Low-level ACU/PTC access	`acu_ptc`
	Optical camera frame view	`optView`
apexdb2	Data reduction software	`class` / `boa`
apexdev	Monitor point DB2 archiver	`apexMonPointArchiver` / `db2*`
	User clients	Misc.
lastarria	Data archive	`db2*`
apexpdc	Project SQL database	mysqld

Table 4.1: This table shows the APECS hosts, typical processes and their shell commands. The standard server processes on `control`, `instruments`, `display` and `abm` are started by `restartAPECSServers` (on `control`). The server processes must run under the `apex` operations account. The recommended observing setup comprises the observer client processes on `pular` or `toco` as started by `restartAPECSObsClients` and the monitoring client processes on `tacora` or `leon` as started using `restartAPECSMonClients`. All client processes must be started under the observing project account in order to get access to macros, catalogs, raw and scientific data, and observing logs.

Host	Applications	Commands
apexcool	Shared memory server	`/etc/init.d/apexCompdata start`
	Temp. sensor. compressor chiller, etc. SCPI parser	`/etc/init.d/apexPt100Server start`
	Synthesizer SCPI parser	`/etc/init.d/apexSynthesizer start`
cs-a1	Calibration unit shared memory setup and SCPI server	`/etc/init.d/calunit start`
	C-cabin mirror server and SCPI server	`/etc/init.d/cmirror start`
	A-cabin optics server and SCPI server	`/home/apex/OpticsA/apexOpticsAControl`
	HET345 control software and SCPI server	`/home/apex/HET345/apexHET345Control`
ffts1	FFTS1 Control Software	`ffts<N>db`
ffts2	FFTS2 Control Software	`ffts<N>db`

Table 4.2: This table shows the embedded system hosts, standard processes and their shell commands.

Symptom	Cause	Solution
`apexAntMount` GUI does not show up after a restart.	Timing of the commands in the startup script did not work due to high CPU or network load. The ABM1 container therefore crashes.	Check CPU load on all APECS central computers ("control", "display" and "instruments"). Check network load (e.g. due to additional web cams which need to be turned off for observing). Restart the system.
CORBA communication fails because the containers were not started correctly. Check the containers on "instruments" using the shell command sequence "`ps waux | grep maciActivate`". A container is not OK if only one thread is visible.	The `maciManager` process was accessed from several clients simultaneously and got overloaded (known problem in ACS 2.0.1). The `maciManager` process then shows more than 3 threads (check on "control" using "`ps waux | grep maciManager`).	Restart the "maciManager" with "maciManagerShutdown / maciManager" (only absolute experts) or restart APECS completely.
Many or all CORBA commands suddenly fail (see "Setting ..." messages in `jlog`).	One or more of the CORBA containers may have crashed. This may be related to the timing problems (see above).	Restart individual containers using the `container<Name>` commands (for experts only) or restart APECS completely.
Data is being recorded into an MBFITS file but there is no online display.	Network delays in the telescope position data stream may cause that no applicable positions are available for the backend data time stamps. The resulting MBFITS file is then unusable. Often these delays are caused by a high CPU load on "control" due to web cam displays or other non-APECS processes. Those must *not* run on "control" !	Check for network delay warning messages in the Online Fits Writer window or use the `comparePosNCTS` script. If there are delays, check the load on "control". Stop any runaway processes or web cam displays that use a lot of CPU time.
Data taking gets very slow or stalls.	The MBFITS writer and/or the online calibrator leak memory and use all the available RAM on `display`.	Check the memory usage on `display` by typing `ps wux | grep apexOnlineFitsWriter` or `ps wux | grep apexOnlineCalibrator` and looking for the %MEM column. Restart the relevant processes if they use more than 50% of the memory. Try to do it by typing <CTRL-C> in the FitsWriter or the calibrator window so that the processes can unregister from CORBA. If this does not work, use `task stop apexOnlineFitsWriter` or `task stop apexOnlineCalibrator`. Restart the processes by typing `apexOnlineFitsWriter` or `apexOnlineCalibrator`.

Table 4.3: Known APECS problems and their solutions.

Symptom	Cause	Solution
Scans get stuck at the end of a subscan.	The MBFITS writer does not return from the CORBA stop command and thus stalls the Observing Engine.	Restart the MBFITS write on `display` by typing <CTRL-C> in the FitsWriter window so that the process can unregister from CORBA. If this does not work, use `task stop apexOnlineFitsWriter` and restart it by typing `apexOnlineFitsWriter`.
Access to all CORBA properties and methods is extremely slow. `getMCPoint` takes 5-10 seconds to return a value but does not throw an exception.	The CORBA Notification Service is slowed down by a dead client such as `jlog` or `apexStatusDisplay`.	Check all APECS computers for such clients, especially within observer accounts where the restart scripts can currently not stop old processes. Stop all those old processes.
Setting or reading instrument parameters fails and scans are canceled.	Most likely the embedded system control software has failed. Very rarely the CORBA containers may crash.	There is an ACS timeout period of 5 seconds when accessing properties. If a `getMCPoint` returns faster than this and fails, then the CORBA object is gone and the container may have crashed. Restart the container. Otherwise the socket communication has timed out (4 seconds). In this case check the corresponding SCPI CID server processes on the APECS machines or on the embedded systems (see tables 4.1 and 4.2) and try restarting the server processes. If this does not help, log on to `instruments` and check the containers as described above.
There is no data in one or more of the MBFITS ARRAYDATA tables.	The TCP connection to one or more backends may have failed.	Check the backend control programs. Check the socket status on port 25144 on `display` and on the backend host. If there are `CLOSE_WAIT` conditions, you may need to wait 10 minutes for Linux to release the resource. In the worst case the backend computers may need to be restarted.
There is no interpolated data in one or more of the MBFITS DATAPAR tables.	The antenna position data published via a Notification Channel may be delayed by more than 10 seconds. Check the delays using the `comparePosNCTS` script.	If there are long delays, then either the load on `control` is too high so that the `Notify_Service` does not get enough time or there is a client that slows down the Notification Channel mechanism by not decoupling the event handling. This condition is difficult to fix unless you know which is the offending client. Try to restart the `apexOnlineFitsWriter` and make sure that no other unnecessary NC clients are running. Check the load on `control`. If all else fails, one needs to restart APECS.

Table 4.4: Known APECS problems and their solutions (ctd.).

Chapter 5

CORBA CDB Reference

Tables 5.1 to 5.3 summarize the names of all APECS CORBA objects as defined in the CDB (Configuration Database). The names are needed to access the objects e.g. via the `apexObsUtils.getMCPoint` command.

CORBA Object Name	Container	Embedded System Host(s)	Process
ABM0:CLOCK	abm0	localhost	
ABM0:TRAJECTORY	abm0	localhost	
ABM0:ANTMOUNT	abm0	localhost	
ABM0:WOBBLER	abm0	localhost	
ABM1:CLOCK	abm1	abm	ActivateStart
ABM1:AMBCAN	abm1	abm	ActivateStart
ABM1:TRAJECTORY	abm1	acu / ptc	N/A
ABM1:ANTMOUNT	abm1	acu	N/A
ABM1:WOBBLER	abm1	wiu	N/A
ABM1:ACU	aux	acu / control	canMonitor
ABM1:PTC	aux	ptc / control	canMonitor
ABM1:PTC:METROLOGY	aux	ptc / control	canMonitor
ABM1:PTC:METROLOGY:TILTMETERS	aux	ptc / control	canMonitor
ABM1:PTC:METROLOGY:TEMPERATURES	aux	ptc / control	canMonitor
ABM1:PTC:SUBREF	aux	ptc / control	canMonitor
ABM1:PTC:SUBREF:HEXAPOD	aux	ptc / control	canMonitor
ABM1:PTC:UPS:1	aux	ptc / control	canMonitor
ABM1:PTC:UPS:2	aux	ptc / control	canMonitor
OPT:CLOCK	opt	instruments	maciActivate opt
OPT:OPTPOINT	opt	instruments	maciActivate opt
APEX:CALUNITS:A	aux	cs-a1	calunit
APEX:CALUNITS:A:MIRRORS	aux	cs-a1	calunit
APEX:CALUNITS:A:COOLER	aux	cs-a1	calunit
APEX:CALUNITS:B	aux	N/A	N/A
APEX:CALUNITS:C	aux	instruments	emuEmbSys

Table 5.1: APECS CORBA Objects and their activators.

CORBA Object Name	Container	Embedded System Host(s)	Process
APEX:ABBA	febe	abba	N/A
APEX:PBE_A	febe	instruments	apexPBEControl
APEX:PBE_B	febe	instruments	apexPBEControl
APEX:PBE_C	febe	instruments	apexPBEControl
APEX:PBE_D	febe	instruments	apexPBEControl
APEX:SZACBE	febe	szbackend	N/A
APEX:SZDCBE	febe	szbackend	N/A
APEX:ARTBE	febe	artemis	N/A
APEX:HOLO	febe	instruments	emuEmbSys
APEX:LABOCA	febe	laboca	N/A
APEX:LABOCA:CALUNIT	febe	laboca	N/A
APEX:LABOCA:AMPLIFIER	febe	laboca	N/A
APEX:LABOCA:POLARIMETER	febe	laboca	N/A
APEX:LABOCA:MAINTENANCE	febe	laboca	N/A
APEX:BOLO350	febe2	bolo350	N/A
APEX:BOLOSZ	febe	bolosz	N/A
APEX:BOLOSZ:FRIDGE	febe	bolosz	N/A
APEX:BOLOSZ:MAINTENANCE	febe	bolosz	N/A
APEX:ARTEMIS450	febe	artemis	N/A
APEX:ARTEMIS200	febe	artemis	N/A
APEX:TILTMETERS:BASE	infra	instruments	apexTiltmeterControl
APEX:RACKCHILLER	infra	apexcool	compdata & pt100server
APEX:COMPRESSORCHILLER	infra	apexcool	compdata & pt100server
APEX:PCCHILLER	infra	apexcool	compdata & pt100server
APEX:COMPRESSOR1	infra	apexcool	compdata & pt100server
APEX:COMPRESSOR2	infra	apexcool	compdata & pt100server
APEX:COMPRESSOR3	infra	apexcool	compdata & pt100server
APEX:COMPRESSOR4	infra	apexcool	compdata & pt100server
APEX:DOORS	infra	apexcool	compdata & pt100server
APEX:HET230	febe2	hfe_control	N/A
APEX:HET345	febe	cs-a1	apexHET345Control
APEX:HET345:MIXER1	febe	cs-a1	apexHET345Control
APEX:HET345:LO1	febe	cs-a1	apexHET345Control
APEX:HET345:LO1:PLL	febe	cs-a1	apexHET345Control
APEX:HET345:LO1:GUNN	febe	cs-a1	apexHET345Control
APEX:HET460	febe2	hfe_control	N/A
APEX:HET690	febe2	hfe_control	N/A
APEX:HET810	febe2	hfe_control	N/A
APEX:HET1500	febe2	hfe_control	N/A
APEX:FLASH460	febe	instruments	emuEmbSys
APEX:FLASH810	febe	instruments	emuEmbSys
APEX:CONDOR	febe	instruments	emuEmbSys
APEX:CHAMP690	febe	champ	N/A
APEX:CHAMP690:DEROTATOR	febe	champ	N/A
APEX:CHAMP690:CALUNIT	febe	champ	N/A
APEX:CHAMP810	febe	champ	N/A
APEX:CHAMP810:DEROTATOR	febe	champ	N/A
APEX:CHAMP810:CALUNIT	febe	champ	N/A

Table 5.2: APECS CORBA Objects and their activators (ctd.).

CORBA Object Name	Container	Embedded System Host(s)	Process
APEX:IF1	aux	if_control / if_proc1	N/A
APEX:IF1:CHAIN1	aux	if_control / if_proc1	N/A
APEX:IF1:CHAIN2	aux	if_control / if_proc1	N/A
APEX:IF2	aux	if_control / if_proc2	N/A
APEX:IF2:CHAIN1	aux	if_control / if_proc2	N/A
APEX:IF2:CHAIN2	aux	if_control / if_proc2	N/A
APEX:IFC1	aux	if_control / champ	N/A
APEX:IFC1:CHAIN1	aux	if_control / champ	N/A
APEX:IFC1:CHAIN2	aux	if_control / champ	N/A
APEX:IFC2	aux	if_control / champ	N/A
APEX:IFC2:CHAIN1	aux	if_control / champ	N/A
APEX:IFC2:CHAIN2	aux	if_control / champ	N/A
APEX:OPTICS:A	aux	cs-a1	apexOpticsAControl
APEX:OPTICS:C	aux	cs-a1	cmirror
APEX:OPTTEL	opttel	instruments	N/A
APEX:RADIOMETER	environ	instruments	apexRadiometerControl
APEX:RADIOMETER:RESULTS	environ	instruments	apexRadiometerControl
APEX:RADIOMETER:MAINTENANCE	environ	instruments	apexRadiometerControl
APEX:FFTS1	febe	ffts1	ffts1<N>db
APEX:FFTS1:BAND1	febe	ffts1	ffts1<N>db
APEX:FFTS1:BAND2	febe	ffts1	ffts1<N>db
APEX:FFTS2	febe	ffts	ffts2<N>db
APEX:FFTS2:BAND1	febe	ffts2	ffts2<N>db
APEX:FFTS2:BAND2	febe	ffts2	ffts2<N>db
APEX:MACS	febe	macs	N/A
APEX:MACS:BAND1	febe	macs	N/A
APEX:MACS:BAND2	febe	macs	N/A
APEX:MACS:BAND3	febe	macs	N/A
APEX:MACS:BAND4	febe	macs	N/A
APEX:MACS:BAND5	febe	macs	N/A
APEX:MACS:BAND6	febe	macs	N/A
APEX:MACS:BAND7	febe	macs	N/A
APEX:MACS:BAND8	febe	macs	N/A
APEX:MACS:MAINTENANCE	febe	macs	N/A
APEX:SYNTHESIZER1	aux	apexcool	compdata & apexSynthesizerControl
APEX:SYNTHESIZER1:DESTINATION	aux	apexcool	compdata & apexSynthesizerControl
APEX:SYNTHESIZER1:FSUNIT	aux	apexcool	compdata & apexSynthesizerControl
APEX:SYNTHESIZER2	aux	apexcool	compdata & apexSynthesizerControl
APEX:SYNTHESIZER2:DESTINATION	aux	apexcool	compdata & apexSynthesizerControl
APEX:SYNTHESIZER2:FSUNIT	aux	apexcool	compdata & apexSynthesizerControl
APEX:REFPLLSYNTHESIZER	aux	apexcool	compdata & apexSynthesizerControl
APEX:REFILLTANK	infra	apexcool	compdata & pt100server
APEX:TEMPERATURES	infra	apexcool	pt100server
APEX:TSGEN	aux	instruments	apexTSGenControl
APEX:WEATHERSTATION	environ	instruments	apexWeatherStationControl

Table 5.3: APECS CORBA Objects and their activators (ctd.).

Bibliography

[1] Hafok, H., Muders, D. & Olberg, M., 2006, APEX SCPI Socket Command Syntax and Backend Data Stream Format, APEX Report APEX-MPI-IFD-0005, Rev. 1.0

[2] Muders, D., 2006, APEX Instruments Generic CORBA IDL Interfaces, APEX Report APEX-MPI-IFD-0004, Rev. 1.5

[3] Muders, D., Hafok, H., Wyrowski, F., Polehampton, E., Belloche, A., König, C., Schaaf, R., Schuller, F., Hatchell, J., v.d.Tak, F., 2006, APECS - The Atacama Pathfinder Experiment Control System, *A&A* Letters, 454, L25-L28

[4] Muders, D., Hatchell, J., Lemke, R., Olberg, M. & Hafok, H., 2002, Software Interfaces for Submillimeter Telescope Instrumentation, APEX Report APEX-IFD-MPI-0001

[5] Muders, D., Polehampton, E. & Hatchell, J., 2006, Multi-beam FITS Raw Data Format, APEX Report APEX-MPI-IFD-0002, Rev. 1.6

[6] Pardo, J. R., Cernicharo, J. & Serabyn, 2001, Atmospheric Transmission at Microwaves (ATM): An Improved Model for Millimeter/Submillimeter Applications, IEEE Trans. on Antennas and Propagation, 49, 1683

[7] Polehampton, E., 2005, APEX Calibrator Manual, APEX Report APEX-MPI-MAN-0012

[8] Rossum, G. v., Drake Jr., F.L., 2001, Python Reference Manual, Release 2.2, http://www.python.org